

Java WS A&A Security Descriptor Framework

Java WS A&A Security Descriptor Framework

Table of Contents

1. Introduction	1
1. Security Descriptor Schemas	1
2. Loading Security Descriptor Files	2
3. Security Descriptor Precedence	2
2. Writing Container Security Descriptor	3
1. Configuring Container Security Descriptor	3
2. Credentials	3
3. Authorization Chain	3
4. Administrator Authorization Chain	4
5. Default Authorization Chain	4
6. Limited proxy policy	5
7. Context lifetime	5
8. Replay attack window	5
9. Context Timer Interval	5
10. Replay Timer Interval	5
11. Trusted Certificates	5
3. Writing Service Security Descriptors	7
1. Configuring Service Security Descriptor	7
2. Credentials	8
3. Authentication methods	8
4. Run-as mode	10
5. Authentication and run-as per-method	11
6. Authorization policy	12
7. Limited proxy policy	12
8. Context lifetime	12
9. Replay attack window	13
4. Writing Resource Security Descriptor	14
1. Configuring Resource Security Descriptor	14
2. Default GridMap	15
3. Credentials	15
4. Authorization chain	15
5. Initializing descriptor	16
5. Common Configuration for server-side security descriptors	17
1. Credentials	17
2. Reject Limited Proxy	18
3. Replay attack prevention	18
4. Context lifetime	19
5. Authorization	19
6. Writing a custom authorization mechanism	22
6. Writing Client Security Descriptors	26
1. Configuring Client Security Descriptor	26
2. Credentials	27
3. Authorization policy	27
4. GSI Secure Conversation	28
5. GSI Secure Message	28
6. GSI Secure Transport	29
7. Username/Password	29
8. Trusted credentials	30
7. Other configuration	31
1. Configuring Default GridMap File	31
Glossary	32

List of Tables

1.1. Security descriptor schema	1
3.1. Authentication methods	9
3.2. Run-as methods	10
5.1. Builtin PDPs	22

Chapter 1. Introduction

Security descriptors contain various security properties like credentials, the *grid map file* location, required authentication and authorization mechanisms and so on. There are four types of security descriptors in the code base for setting container, service, resource and client security properties:

Descriptor Type	Functionality
container security descriptor	determines the container level security requirement that needs to be enforced.
service security descriptor	determines the service level security requirement that needs to be enforced.
resource security descriptor	determines the resource level security requirement that needs to be enforced.
client security descriptor	determines the security properties that need to be used for a particular invocation.

The security descriptors (container, security and resource) can be created and altered programmatically (as opposed to writing a security descriptor file). For the service and container descriptor, we recommend writing a security descriptor file so that the security properties are initialized at start up.

Service and container security descriptors can be configured as XML files in the global and service deployment descriptor as shown below. Resource security descriptors can only be created dynamically, either programmatically or from a descriptor file. Client security descriptor can be configured as a XML file and set as property on Stub.

1. Security Descriptor Schemas

All security descriptor files need to comply with a defined schema and should be written within the defined namespace.

Table 1.1. Security descriptor schema

Descriptor	Schema	Namespace	Root Element
Container security descriptor	Schema ¹	http://www.globus.org/security/descriptor/container	containerSecurityConfig
Service security descriptor	Schema ²	http://www.globus.org/security/descriptor/service	serviceSecurityConfig
Resource security descriptor	Schema ³ , (Same schema as service)	http://www.globus.org/security/descriptor/service	serviceSecurityConfig
Client security descriptor	Schema ⁴	http://www.globus.org/security/descriptor/client	clientSecurityConfig

¹ http://viewcvs.globus.org/viewcvs.cgi/wsrif/schema/core/security/descriptor/container_security_descriptor.xsd?view=annotate&pathrev=globus_4_2_branch

² http://viewcvs.globus.org/viewcvs.cgi/wsrif/schema/core/security/descriptor/service_security_descriptor.xsd?view=annotate&pathrev=globus_4_2_branch

³ http://viewcvs.globus.org/viewcvs.cgi/wsrif/schema/core/security/descriptor/service_security_descriptor.xsd?view=annotate&pathrev=globus_4_2_branch

⁴ http://viewcvs.globus.org/viewcvs.cgi/wsrif/schema/core/security/descriptor/client_security_descriptor.xsd?view=annotate&pathrev=globus_4_2_branch

2. Loading Security Descriptor Files

If a security descriptor is configured to be read from a file, it is loaded as follows:

1. As a file if an absolute file path is specified.
2. As a resource (can be included as part of jar file).
3. As a file, assuming that the specified path is relative to the installation root, typically pointed to by the environment variable `GLOBUS_LOCATION`.

If the security descriptor file is altered at runtime, it will **not** be reloaded

3. Security Descriptor Precedence

If security properties are configured in multiple locations, then the following order of precedence is used

1. Resource security descriptor
2. Service security descriptor
3. Container security descriptor

Chapter 2. Writing Container Security Descriptor

1. Configuring Container Security Descriptor

This section describes configuration of container security descriptor.

1. The container security descriptor can be configured in the `<globalConfiguration>` section of the Java WS Core deployment descriptor. That file is in `wsrf/java/core/source/deploy-server.wsdd` if editing the source, prior to deploying, or `$GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd` in a binary install.

```
<globalConfiguration>
...
<parameter name="containerSecDesc"
value="/path/to/container/descriptor/file.xml">
...
</globalConfiguration>
...
```

2. The descriptor file name can also be specified as a parameter when the Java WS Core container is started up. The option is **`-containerSecDesc "/path/to/container/descriptor/file.xml"`**



Note

This setting takes precedence over 1

3. This is represented by `org.globus.wsrf.impl.security.descriptor.ContainerSecurityDescriptor` .

If a container security descriptor file is configured as described in [Section 1, “Configuring Container Security Descriptor”](#), then an object is created and stored. To alter the values, use the API provided in `org.globus.wsrf.impl.security.descriptor.ContainerSecurityConfig` .

This is useful to configure containers that are started up for notifications. An instance of `ContainerSecurityDescriptor` object can be set as property `org.globus.wsrf.container.CONTAINER_DESCRIPTOR`.

2. Credentials

To configure container level credentials, refer to [Section 1, “Credentials”](#)

3. Authorization Chain

To configure authorization at container level for use if not overridden by service or resource level policy, refer to [Section 5, “Authorization”](#)

4. Administrator Authorization Chain

Other than the container/service/resource authorization, an administrative-level authorization chain can be configured using the `<adminAuthz>` element. The decision returned by this chain overrides subsequent authorization decision. That is, if the administrator's authorization chain returns a deny, the rest of the configured authorization (at container/service/resource) is *not* evaluated and the operation is denied. If the administrator's chain returns the permit, the rest of the configuration is evaluated to see if the operation is allowed.

The element has the same schema as described in [Section 5, “Authorization”](#), with the outer element called `adminAuthz` in place of `authzChain`.

Example:

```
<containerSecurityConfig
xmlns="http://www.globus.org/security/descriptor/container">
...
<adminAuthz>

  <pips>
<interceptor name="scope2:org.globus.sample.PIP1"/>
</pips>
  <pdps>
<interceptor name="fool:org.foo.authzMechanism
bar1:org.bar.barMechanism"/>
</pdps>

</adminAuthz>
...
</containerSecurityConfig/>
```

5. Default Authorization Chain

This element is used to configure default properties for any interceptor configured in authorization chains. The schema for this is similar to the authorization chain specification as described in [Section 5, “Authorization”](#) and allows for `xsd:any` as the interceptor parameter.

```
<defaultAuthzParam>
<interceptor name="scope1:org.globus.sample.SamplePDP"/>
<parameter>
<param:nameValueParam>
<param:parameter name="policy-file"
value="/home/user1/samplePDPCConfig"/>
</param:nameValueParam>
</parameter>
</interceptor>
</defaultAuthzParam>
```

6. Limited proxy polocy

Container can choose to require that clients use full proxies for access and reject limited proxies. To configure such policy, refer to [Section 2, “Reject Limited Proxy”](#)

7. Context lifetime

You can control the lifetime of the context with GSI Secure Conversation as an authentication mechanism, as described in [Section 4, “Context lifetime”](#).

8. Replay attack window

You can control the replay attack window for services that allow for GSI Secure Message, as desribed in [Section 3, “Replay attack prevention”](#).

9. Context Timer Interval

When *GSI Secure Conversation* is used, a security context is established and a worker thread cleans up expired contexts. This parameter sets the interval on the timer thread that collects expired contexts established when GSI Secure Conversation is used. The value is the number of seconds between each run and defaults to 10 minutes.

```
<containerSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/container">
  ...
  <context-timer-interval value="100000"/>
  ...
</containerSecurityConfig>
```

10. Replay Timer Interval

This parameter sets the interval on the timer thread that collects expired message digest ids, stored to prevent replay attack in the case of Secure Message. The value is set in seconds and the default value is 1 minute.

```
<containerSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/container">
  ...
  <replay-timer-interval value="100"/>
  ...
</containerSecurityConfig>
```

11. Trusted Certificates

This parameter sets the location of trusted certificates to be used. The value should be a comma-separated list of locations.

```
<containerSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/container">
  ...
  <trusted-certificates value="/home/user1/trustedCerts
/home/user1/newCerts"/>
  ...
</containerSecurityConfig>
```

```
...  
</containerSecurityConfig>
```

If this configuration is not set, the underlying CoG JGlobus library is used to pick up trusted certificates. The library attempts to load the certificates as described in [Section 1, “Trusted Certificates Location”](#) .

Chapter 3. Writing Service Security Descriptors

All security properties of a service can be configured using a descriptor file. Each section in this chapter explains the various properties and how they can be configured.

The following section explains how a descriptor can be configured at the service level.

1. Configuring Service Security Descriptor

1. The service security descriptor can be configured in the service's deployment descriptor section as a parameter. The parameter is a name/value that provides the path to the security descriptor file.

```
<service name="MyDummyService"
provider="Handler"
style="document">
...
<parameter name="securityDescriptor"
value="org/globus/wsrfl/impl/security/descriptor/security-co
...
</service>
```

2. A `org.globus.wsrfl.impl.security.descriptor.ServiceSecurityDescriptor` object can be created and initialized in the service's constructor.

```
public class MyDummyService {
public MyDummyService() throws Exception {

ServiceSecurityDescriptor serviceDesc =
new ServiceSecurityDescriptor();

// set security properties on the above object
using get/set methods
// in the API

ServiceSecurityHelper
.setSecurityDescriptor("DummyServiceName"
serviceDesc);
}
}
```



Note

This method takes precedence over 1

3. A `ServiceSecurityDescriptor` object can be created similar to above, but initialized from a file and set in the constructor.

```
public class MyDummyService {  
  
    public MyDummyService() throws Exception {  
  
        ServiceSecurityDescriptor serviceDesc =  
            new ServiceSecurityDescriptor("/path/to/security/file");  
  
        ServiceSecurityHelper  
            .setSecurityDescriptor("DummyServiceName"  
                serviceDesc);  
    }  
}
```



Note

This method takes precedence over 1

Programmatic creation and altering of a service security descriptor is not discussed in detail here. Refer to [????](#) for some details on using the API to modify a descriptor.

2. Credentials

To configure credentials specific to your service rather than use the container level configuration, refer to [Section 1, “Credentials”](#)

3. Authentication methods

This section describes configuring policy on the authentication mechanism required by your service. This specifies the security policy that determines the authentication methods clients should use for contacting your service.

The authentication method required for accessing a service can be configured in the descriptor using the `<auth-method>` element. A per method configuration can also be done as described in [Section 5, “Authentication and run-as per-method”](#).

Currently, the following authentication methods are supported:

Table 3.1. Authentication methods

Authentication Method	Element	Options/Notes
No Authentication	<none/>	This method <i>cannot</i> be specified with any other authentication method.
GSI Secure Message	<GSISecureMessage/>	The <protection-level> sub element can be used to specify a protection level that must be applied to the message: <ul style="list-style-type: none"> • <integrity/> , indicates that the message must be integrity-protected (signed). • <privacy/> , indicates that the message must be privacy-protected (encrypted and signed).
GSI Secure Conversation	<GSISecureConversation/>	The <protection-level> sub element can be used to specify a protection level that must be applied to the message: <ul style="list-style-type: none"> • <integrity/> , indicates that the message must be integrity-protected (signed). • <privacy/> , indicates that the message must be privacy-protected (encrypted and signed).
GSI Secure Transport Authentication	<GSITransport/>	The <protection-level> sub element can be used to specify a protection level that must be applied to the message: <ul style="list-style-type: none"> • <integrity/> , indicates that the message must be integrity-protected (signed). • <privacy/> , indicates that the message must be privacy-protected (encrypted and signed).

Notes:

- Multiple authentication methods can be specified under the <auth-method> element (except for the <none/> method, see above). As long as one of the specified authentication methods is used, access to the service is allowed.
- If multiple authentication methods are specified, they need to be in alphabetical order. That is, the following order needs to be maintained: 1) GSISecureConversation, 2) GSISecureMessage, 3) GSISecureTransport. This does not imply that all three need to be specified, but that the specified authentication mechanisms need to comply with the above order.
- If *no* <protection-level> sub-element is specified, then all protection levels are available to clients. However, if the <protection-level> sub-element *is* specified, then the service will only accept the protection levels listed under said element.
- The `org.globus.wsrfl.impl.security.authentication.SecurityPolicyHandler` handler *must* be installed properly in order for this to work. This handler is installed by default.
- If a security descriptor is *not* specified, authentication method enforcement is *not* performed.

Example:

```
<serviceSecurityConfig xmlns="http://www.globus.org/security/descriptor/service">
    <!-- default auth-method for any other method -->
    <auth-method>
    <GSISecureConversation/>
    </auth-method>
</securityConfig>
```

4. Run-as mode

This section describes the credentials your service should use for the operation being invoked. This is important if remote secure operations are being invoked from your service, since it determines what credentials are used for such an access.

The `<run-as>` element is used to configure the JAAS run-as identity under which the service method will be executed. The run-as identity can be configured on a per method basis also as described in [Section 5, “Authentication and run-as per-method”](#). Currently, the following run-as identities are supported:

Table 3.2. Run-as methods

Element	Functionality
<code><run-as value="caller" /></code>	<p>The service method will be run with the security identity of the client. The caller Subject will contain the following:</p> <ul style="list-style-type: none"> • If using <i>GSI Secure Message</i>: a <code>GlobusPrincipal</code> (the identity of the signer) is added to the principal set of the caller-identity Subject. Also, the signer's certificate chain is added to the public credentials set of the Subject object. • If using <i>GSI Secure Conversation</i>: a <code>GlobusPrincipal</code> (the identity of the initiator) is added to the principal set of the Subject. <ul style="list-style-type: none"> • If client authentication was performed, the client's certificate chain will be added to the public credentials set of the Subject object. • Also, if delegation was performed, the delegated credential is added to the private credential set of the Subject object. • If grid map file authorization was performed, a <code>UserNamePrincipal</code> is added to the principal set of the Subject object.
<code><run-as value="service" /></code>	The service method will be run with the security identity of the service itself (if the service has one, otherwise the container identity will be used).
<code><run-as value="resource" /></code>	The service method will be run with the security identity of the resource. If no resource is specified or if the resource does not have a configured subject, credentials in this order of occurrence will be used: service credential, container credential.
<code><run-as value="system" /></code>	The service method will be run with the security identity of the container.

Notes:

- *resource-identity* is the default setting.

- The `org.globus.wsrfl.impl.security.authentication.SecurityPolicyHandler` handler *must* be installed properly in order for this to work. It is installed by default.
- If the security descriptor is *not* specified, then the run-as identity is not set and there will be no JAAS subject associated with the execution of the operation. This means that any method calls that require credentials and that are invoked by the service method itself will fail.

Example:

```
<serviceSecurityConfig xmlns="http://www.globus.org/security/descriptor/service">

    <!-- default run-as for any other method -->
    <run-as>
    <service-identity/>
    </run-as>

</serviceSecurityConfig>
```

5. Authentication and run-as per-method

This section describes configuring the authentication and run-as mechanism per method, rather than for the whole service.

A per-method configuration can be used to define expected authentication methods for the method and also a run-as configuration. The element `methodAuthentication` is used to list all the method configuration. For each method, element `method` with method name as attribute needs to be used. The method name attribute can either be the local name of the method or a string representation of the QName of the method. If the QName is used, the `toString()` representation from the `javax.xml.namespace.QName.toString()`, should be used. For example, `{http://foo.bar}methodName`. If a method does not have such a configuration, the default configuration, as described in the last two sections, is used.

Example

```
<serviceSecurityConfig
xmlns="http://www.globus.org/security/descriptor/service">
...
<method name="findServiceData">
<auth-method>
<none/>
</auth-method>
</method>

<method name="{http://foo.bar}subtract">
<run-as>
<service-identity/>
</run-as>
</method>

<method name="destroy">
<auth-method>
<GSISecureMessage/>
<GSISecureConversation>
```

```
<protection-level>
<integrity/>
</protection-level>
</GSI SecureConversation>
</auth-method>
</method>

<!-- default run-as for any other method -->
<run-as>
<system-identity/>
</run-as>

<!-- default auth-method for any other method -->
<auth-method>
<GSI SecureConversation/>
</auth-method>
...
</serviceSecurityConfig>
```

In the above example:

- the `findServiceData()` operation does not require any authentication. Since no run-as per method is specified, default specification, system-identity is used.
- the `destroy()` operation requires either *GSI Secure Message* authentication with either level of protection or *GSI Secure Conversation* authentication with integrity protection. Default run-as is used.
- `substract` method does not have a specific authentication specified, so the default GSI Secure Conversation is used. But the operation is run with service identity.
- all other operations must be authenticated with *GSI Secure Conversation* with either level of protection.

6. Authorization policy

To configure authorization policy at service level, refer to [Section 5, “Authorization”](#). In addition to the authorization mechanisms shipped, you can write custom authorization mechanism for a service as described in [Section 6, “Writing a custom authorization mechanism”](#)

7. Limited proxy policy

Your service can choose to require that clients use full proxies for access and reject limited proxies. To configure such policy, refer to [Section 2, “Reject Limited Proxy”](#)

8. Context lifetime

If your service allows for GSI Secure Conversation as an authentication mechanism, you can control the lifetime of the context as described in [Section 4, “Context lifetime”](#).

9. Replay attack window

If your service allows for GSI Secure Message, you can control the replay attack window at the service level as described in [Section 3, “Replay attack prevention”](#).

Chapter 4. Writing Resource Security Descriptor

1. Configuring Resource Security Descriptor

Secure resources must implement the `org.globus.wsrp.security.SecureResource` interface.

1. A `ResourceSecurityDescriptor` object can be created and initialized in the resource's constructor. The object should be returned as a part of the `getSecurityDescriptor` method.

```
public MyDummyResource implements SecureResource
{

    private ResourceSecurityDescriptor desc = null;

    public MyDummyResource() throws Exception {

        this.desc = new ResourceSecurityDescriptor();

        // set security properties on the above object
        using get/set methods
        // in the API
    }

    public ResourceSecurityDescriptor
    getSecurityDescriptor() {
        return this.desc;
    }
}
```

2. A `ResourceSecurityDescriptor` object can be created similar to above, but initialized from a file and set in the constructor.

```
public MyDummyResource implements SecureResource
{

    private ResourceSecurityDescriptor desc = null;

    this.desc = new ResourceSecurityDescriptor("/path/to/secure")
    this.desc.initialize();
}

public ResourceSecurityDescriptor
getSecurityDescriptor() {
    return this.desc;
}
```

```
}
```

Resource level security can be set up using a resource security descriptor. The configuration properties and schema are identical to the service security descriptor. A resource security descriptor overrides any service or container level security settings. To make a resource secure, it needs to implement `org.globus.wsrfl.impl.security.SecureResource`. This interface has a method that returns an instance of `org.globus.wsrfl.impl.security.descriptor.ResourceSecurityDescriptor`. If null is returned, it is assumed that no security is set on the resource.

A resource security descriptor object can be created by reading settings from a descriptor file. The file needs to be written as described in [Chapter 5, Common Configuration for server-side security descriptors](#). Refer to [Section 1, “Configuring Resource Security Descriptor”](#) for information on configuring resource security descriptors.

The Resource security descriptor object also exposes an API to set and get all properties that are described in [Chapter 5, Common Configuration for server-side security descriptors](#). The descriptor can be set up programmatically using the API.

Examples:

The following code snippet creates a resource descriptor object directly:

```
ResourceSecurityDescriptor desc = new
    ResourceSecurityDescriptor();
desc.setRejectLimitedProxy("true");
```

2. Default GridMap

To set default gridmap:

```
GridMap gridmap = new GridMap();
// set GridMap mappings
desc.setDefaultGridMap(gridmap);
```

3. Credentials

To set credentials:

```
import javax.security.auth.Subject;

Subject subject = new Subject();
// set public/private credentials on subject object
desc.setSubject(subject);
```

4. Authorization chain

To set up an authorization chain:

```
String[] pdpChain = new String[2];
pdpChain[0] = "prefix1:some.PDP";
pdpChain[1] = "prefix2:some.other.PDP";

// Similar API for PIPs and Bootstrap PIP
String[] pips = new String[1];
pips[0] = "pip1:some.PIP";

ResourcePropertiesChainConfig config = new
ResourcePropertiesChainConfig();
config.setProperty("prefix1", "property1", value1);

AuthorizationEngine engine =
AuthzUtil.getAuthzEngine("Chain name", pips, pdpChain,
config);
desc.setAuthzEngine(engine);
```

5. Initializing descriptor

To force initializing of a resource security descriptor:

```
desc.setInitialized(false);
desc.initialize();
```

Chapter 5. Common Configuration for server-side security descriptors

The next few sections deal with writing server-side security descriptor files, that is, container, service and resource descriptor files to set various properties. Only the properties that are common to all these descriptors are discussed here. Other properties specific to each descriptor are discussed in sections specific to the container/service or resource descriptor.

When parameters are configured in multiple descriptors the order of precedence as described in [Section 3, “Security Descriptor Precedence”](#)

1. Credentials

The container and each service can each be configured with a separate set of credentials. The credentials can be set using either: a) the path to a proxy file, or b) the path to a certificate and key file. If the configured credential file is modified/updated at runtime, the credentials will be automatically reloaded. The credentials can be configured by adding one of the following blocks to the container or service security descriptor.

Example for option (a):

```
<serviceSecurityConfig xmlns="http://www.globus.org/security/descriptor/service">
    ...
    <proxy-file value="proxyFile"/>
    ...
</serviceSecurityConfig>
```

Example for option (b):

```
<serviceSecurityConfig
    xmlns="http://www.globus.org/security/descriptor/service">
    ...
    <credential>
    <cert-key-files>
    <key-file value="keyFile"/>
    <cert-file value="certFile"/>
    </cert-key-files>
    </credential>
    ...
</serviceSecurityConfig>
```



Note

The above examples show use of service security descriptor. If setting in container security descriptor, set the namespace and outer element as shown in [Section 1, “Security Descriptor Schemas”](#).

Credentials can be configured at resource, service or container level and the framework will look for credentials in the following order:

1. Resource credentials
2. *Service credentials*

3. Container credentials
4. Default credentials. If credentials are not configured using any of the above methods, then the underlying CoG JGlobus library is used. This will attempt to load the *proxy certificate* of the user that is running the container as described in [Section 5, “Proxy file Location”](#).

2. Reject Limited Proxy

This parameter can be used to configure if clients that present limited proxies can be allowed to authenticate successfully. By default, limited proxies are accepted.

```
<serviceSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/service">
  ...
  <reject-limited-proxy value="true"/>
  ...
</serviceSecurityConfig>
```



Note

The above examples show use of service security descriptor. If setting in the container security descriptor, set the namespace and outer element as shown in [Section 1, “Security Descriptor Schemas”](#).

3. Replay attack prevention

For message-level security, one may also set the amount of time for which to track received messages for the purpose of preventing replay attacks. Messages outside of this window will be rejected automatically, whereas messages within this window are checked against recently received messages through the use of the message UUID.

- Parameter *replay-attack-filter* can be set to true or false to enable or disable replay attack prevention framework. By default, this feature is enabled.

```
<serviceSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/service">
  ...
  <replay-attack-filter value="true"/>
  ...
</serviceSecurityConfig>
```

- Parameter *replay-attack-window* can be set to the number of minutes the replay window should be. By default it is 5 minutes.

```
<serviceSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/service">
  ...
  <replay-attack-window value="100"/>
  ...
</serviceSecurityConfig>
```

 **Note**

The above examples show use of the service security descriptor. If setting in the container security descriptor, set the namespace and outer element as shown in [Section 1, “Security Descriptor Schemas”](#).

4. Context lifetime

When *GSI Secure Conversation* is used, a security context is established and, by default, the life of the context is determined by the least lifetime of the chain of certificates used in establishing the context. The value of the lifetime can be altered to be lesser than the above value by setting the value for following parameter in milliseconds.

```
<serviceSecurityConfig
  xmlns="http://www.globus.org/security/descriptor/service">
  ...
  <context-lifetime value="1000"/>
  ...
</serviceSecurityConfig>
```

 **Note**

The above examples show use of service security descriptor. If setting in container security descriptor, set namespace and outer element as shown in [Section 1, “Security Descriptor Schemas”](#).

5. Authorization

The container and each service/resource can be configured with a chain of interceptors, where each interceptor is a Policy Decision Point (PDP) or Policy Information Point (PIP). The element `authzChain` can be used to configure this.

- Each chain can contain an optional list of Bootstrap PIPs, an optional list of PIPs and a list of PDPs (with at least one PDP).
- Each interceptor name is scoped and the format is *prefix:FQDN of the interceptor*. For example, *self:org.globus.ws-rf.impl.security.authorization.SelfAuthorization*. The prefix is used to allow multiple instances of the same interceptor to exist in the same PDP chain.

Example:

```
<serviceSecurityConfig
xmlns="http://www.globus.org/security/descriptor/service">
...
<authzChain>

  <pips>
  <interceptor name="scope2:org.globus.sample.PIP1"/>
  </pips>
  <pdps>
  <interceptor name="fool:org.foo.authzMechanism
bar1:org.bar.barMechanism"/>
  </pdps>

</authzChain>
```

```
...  
<serviceSecurityConfig/>
```

- Bootstrap PIPs are optional and by default, *org.globus.wsrfl.impl.security.authorization.X509BootstrapPIP* is used by the framework. Any other PIPs listed within the `<bootstrapPips>` element is appended to the default PIP. If the configuration should override it (that is, *X5008BootstrapPIP* should not be used), the optional attribute `overwrite` can be set to `true`.

```
<serviceSecurityConfig  
xmlns="http://www.globus.org/security/descriptor/service">  
...  
  
<authzChain combiningAlg="org.globus.sample.SampleAlg">  
<bootstrapPips overwrite="true">  
<interceptor name="scope1:org.globus.sample.BootstrapPIP1"/>  
</bootstrapPips>  
<pips>  
<interceptor name="scope2:org.globus.sample.PIP1"/>  
</pips>  
<pdps>  
<interceptor name="scope3:org.globus.sample.PDP1"/>  
</pdps>  
</authzChain>  
..  
</serviceSecurityConfig>
```

In the above case, *X509BootstrapPIP* will not be used and the *BootstrapPIP1* will be used.

```
<serviceSecurityConfig  
xmlns="http://www.globus.org/security/descriptor/service">  
...  
  
<authzChain combiningAlg="org.globus.sample.SampleAlg">  
<bootstrapPips>  
<interceptor name="scope1:org.globus.sample.BootstrapPIP1"/>  
</bootstrapPips>  
<pips>  
<interceptor name="scope2:org.globus.sample.PIP1"/>  
</pips>  
<pdps>  
<interceptor name="scope3:org.globus.sample.PDP1"/>  
</pdps>  
</authzChain>  
..  
</serviceSecurityConfig>
```

In the above case, *X509BootstrapPIP* followed by *BootstrapPIP1* will be used.

- The authorization chain can be configured with a combining algorithm using the attribute `combiningAlg`. The value should be a FQDN of a class that implements `org.globus.wsrfl.security.authorization.Author-`

izationEngineSpi . The attribute is optional and by default the org.globus.wsrfl.impl.security.authorization.providers.PermitOverrideAlgorithm is used.

Example:

```
<serviceSecurityConfig
xmlns="http://www.globus.org/security/descriptor/service">
...

<authzChain combiningAlg="org.globus.sample.SampleAlg">
<pips>
<interceptor name="scope2:org.globus.sample.PIP1"/>
</pips>
<pdps>
<interceptor name="scope3:org.globus.sample.PDP1"/>
<interceptor name="scope4:org.globus.sample.PDP2"/>
</pdps>
</authzChain>
..
</serviceSecurityConfig>
```

In the above, the default X509BootstrapPIP will be used. Following that, the PIPs, PIP1, will be invoked to collect attributes. Finally the SampleAlg combining algorithm with the configured PDPs (PDP1 and PDP2) are run to determine the decision.

- Each interceptor can specify a parameter value and the schema defines it as xsd:any to allow for any user defined parameters. The parser extracts the elements in the <parameter> element and returns them as a DOM Element. It is left up to the interceptor to parse the element. The DOM object created is placed in the ChainConfig object passed to the authorization engine as a parameter called "parameterObject". The prefix will be the scope specified in the interceptor name.

Since schema validation is done, a schema must be supplied for the user-defined parameters. The schema location is loaded as a resource and hence can be included in some jar placed in the GLOBUS_LOCATION lib directory.

- The toolkit provides a parameter schema by default that allows for a name/value pair, where the value is a string.
Example :

```
<containerSecurityConfig xmlns="http://www.globus.org/se
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.globus.org/security/descr
name_value_type.xsd"
xmlns:param="http://www.globus.org/security/descriptor">
<authzChain>
<pdps>
<interceptor name="gridmapAuthz:org.globus.wsrfl.impl.sec
<parameter>
<param:nameValueParam>
<param:parameter name="gridmap-file"
value="/home/user1/grid-mapfile"/>
</param:nameValueParam>
</parameter>
</interceptor>
```

```

</pdps>
</authzChain>
</containerSecurityConfig>

```

When the above is parsed, a DOM Element is constructed with element `<param:nameValuParam>` and stored in the ChainConfig object as parameter with name "gridmapAuthz:parameterObject". The GridMap PDP, uses ObjectDeserializer to retrieve the name/value pairs.

All PDPs and PIPs shipped with the toolkit are provided here: [PIP Reference](#) and [PDP Reference](#).

Following PDPs are a part of the toolkit and are configured using short tags. The framework maps and plugs in the scoped name of the PDP at the time of authorization.

Table 5.1. Builtin PDPs

Default Descriptor Configuration	Default Prefix	Reference
acl	aclAuthz	Access Control List PDP
none	noneAuthz	NoAuthorization PDP
self	selfAuthz	Self Authorization PDP
gridmap	gridmapAuthz	GridMapAuthorization PDP
identity	idenAuthz	IdentityAuthorization PDP
host	hostAuthz	HostAuthorization PDP
samlCallout	samlAuthz	SAMLAuthorizationCallout PDP
userName	userNameAuthz	Username Authz PDP
samlAssertion	samlAssertionAuthz	SAML Authz Assertion PDP

Other than these, any custom authorization scheme could be configured with its own configuration information. Refer to [Section 6, "Writing a custom authorization mechanism"](#), for details on writing a custom authorization mechanism.

6. Writing a custom authorization mechanism

The authorization handler can be configured to call out to a custom PIP or PDP. The custom PDP class must implement the interface `org.globus.wsrp.security.PDP` and the custom PIP class must implement the interface `org.globus.wsrp.security.PIP`.

Example PIP:

```

package org.foobar;

import ....;

public class FooPIP implements PIP
{

    public String CURRENT_TIME = "org.foobar.current.time";

    public void collectAttributes(Subject peerSubject,

```

```
MessageContext context,
QName operation) throws AttributeException {

    // collect attributes, say attributes from certificate
    extension
    // store in message context with some property.
    // Example here stores current time.
    messageContext.setProperty(FooPIP.CURRENT_TIME,
    Calendar.getInstance());
}

public void initialize(PDPConfig config,
String name,
String id)
throws InitializeException {

    /* Read the initialization information
    */

}

public void close() throws CloseException {
this. authorizedIdentity = null;
}
}
```

To use the above PIP one would configure a service security descriptor with the following authorization settings:

```
<securityConfig xmlns="http://www.globus.org">
    ...
    <authz value="fool:org.foobar.FooPIP"/>
    ...
</securityConfig/>
```

Example PDP:

```
package org.foobar;

import ....;

public class FooPDP implements PDP
{
private Principal authorizedIdentity;

    /* Not used by the current code */
public String[] getPolicyNames() {
return new String[0];
}

    /* Not used by the current code */
public Node getPolicy(Node query)
throws InvalidPolicyException {
return null;
}
```

```
}

/* Not used by the current code */
public Node setPolicy(Node policy)
throws InvalidPolicyException {
return null;
}

public boolean isPermitted(Subject peerSubject,
MessageContext context,
QName operation)
throws AuthorizationException {

// The parameters set by FooPIP can be accessed here.
Calendar currentTime =
(Calendar)context.getProperty(FooPIP.CURRENT_TIME);

if (peerSubject == null) {
return false;
}

Set peerPrincipals = peerSubject.getPrincipals();

if ((peerPrincipals == null) ||
peerPrincipals.isEmpty()) {
return false;
}

/* Check if the peer identity and the authorized
* identity match
*/

return peerPrincipals.contains(this.authorizedIdentity);
}

public void initialize(PDPCConfig config,
String name,
String id)
throws InitializeException {

/* Read the initialization information from the service
* specific WSDD parameter <name>-authorizedIdentity
*/

this.authorizedIdentity =
new GlobusPrincipal((String) config.getProperty(
name, "authorizedIdentity"));
}

public void close() throws CloseException {
this.authorizedIdentity = null;
}
}
```

To use the above PDP one would configure a service security descriptor with the following authorization settings:

```
<securityConfig xmlns="http://www.globus.org">
  ...
  <authz value="foo1:org.foobar.FooPDP"/>
  ...
</securityConfig/>
```

This security descriptor (identified as `.../foo-pdp-security-config.xml` below) can then be used by a service. The association is created by adding a couple of parameters to the service's WSDD entry:

```
...
  <service name="MyDummyService"
    provider="Handler"
    style="document">
    ...
    <parameter name="securityDescriptor"
      value=".../foo-pdp-security-config.xml"/>
    <parameter name="foo1-authorizedIdentity"
      value="/DC=org/DC=doe/OU=People/CN=John D"/>
    ...
  </service>
```

Note that the parameter `foo1-authorizedIdentity` in the above configures the identity the PDP uses for authorizing incoming requests. The parameter name is derived by composing the prefix (`foo1`) used when specifying the PDP in the security descriptor with the property (`authorizedIdentity`) used in the PDP code.

Chapter 6. Writing Client Security Descriptors

1. Configuring Client Security Descriptor

1. Client security descriptors from a file can be configured directly on the stub as follows:

```
// Client security descriptor file
String CLIENT_DESC =
"org/globus/wsrp/samples/counter/client/client-security-con
//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR_FILE,
CLIENT_DESC);
```

2. Client security descriptors object can be constructed from a file and configured directly on the stub as follows:

```
// Client security descriptor file
String CLIENT_DESC = "org/globus/wsrp/samples/counter/client
ClientSecurityDescriptor desc = new
ClientSecurityDescriptor(CLIENT_DESC);
//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR,
desc);
```



Note

This takes precedence over 1

3. A client security descriptors object can be created and get/set methods can be used to set security properties. The object can then be configured on the stub as follows:

```
ClientSecurityDescriptor desc = new
ClientSecurityDescriptor();

// set security properties on the above object
using set/get object

//Set descriptor on Stub
((Stub)port)._setProperty(Constants.CLIENT_DESCRIPTOR,
desc);
```

To initialize the descriptor, use the API in `org.globus.wsrp.impl.security.descriptor.ClientSecurityConfig`.

**Note**

This takes precedence over 1

2. Credentials

The client can be configured with credentials using the descriptor. The credentials can be set using either: a) the path to a proxy file, or b) the path to a certificate and key file. The credentials can be configured by adding one of the following blocks to the client security descriptor.

Example for option (a):

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor/client">
    ...
    <proxy-file value="proxyFile"/>
    ...
</clientSecurityConfig>
```

Example for option (b):

```
<clientSecurityConfig
    xmlns="http://www.globus.org/security/descriptor/client">
    ...
    <credential>
    <cert-key-files>
    <key-file value="keyFile"/>
    <cert-file value="certFile"/>
    </cert-key-files>
    </credential>
    ...
</clientSecurityConfig>
```

If credentials are not configured using any of the above methods, then the underlying CoG JGlobus library is used. This will attempt to load the *proxy certificate* of the user that is running the container as described in [Section 5, “Proxy file Location”](#).

3. Authorization policy

The `<authz>` element is used to determine the mechanism to use to authorize the server that is being contacted. Note that the security descriptor cannot be used to configure custom client authorization. Refer to [Chapter 10, Authorization domain-level interface](#) for details. The following values are currently supported:

Configuration	Functionality
none	No authorization is done.
self	Self authorization is done, i.e the server should be running with the same credentials as the client.
host	Host authorization is done, i.e the server should be running with credentials that have the host name it is running on embedded in it.
hostSelf	Host authorization is done (i.e the server should be running with credentials that have the host name it is running on embedded in it). If that fails, an attempt at self

	authorization (i.e the server should be running with same credentials as client) is made.
<i>any other string</i>	Identity authorization is done using the value as the identity, i.e the server should be running with identity specified as value.

The following sample configures self authorization:

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor"
...
<authz value="self"/>
...
</clientSecurityConfig>
```

4. GSI Secure Conversation

The client can be configured to do GSI Secure Conversation using the element `<GSI SecureConversation>`. The following subelements can be used to set various properties

Element	Functionality
<code><integrity></code>	Sets protection level to signature.
<code><privacy></code>	Sets protection level to encryption (signature is also done).
<code><anonymous></code>	Server is accessed as anonymous.
<code><delegation value=" type of delegation " ></code>	Determines the type of delegation to be done. The value can be set to full or limited. If the <i>delegation</i> element is not used, no delegation is done. If delegation is enabled, some form of client authorization is required.
<code><context-lifetime></code>	Determines the lifetime of the context established. If not specified, the least lifetime of the chain of certificates used in establishing the context is used as context lifetime.

The following sample sets GSI Secure Conversation with privacy and full delegation:

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor"
...
<GSI SecureConversation>
<privacy/>
<delegation value="full"/>
</GSI SecureConversation>
...
</clientSecurityConfig>
```

5. GSI Secure Message

The client can be configured to do GSI Secure Message using the element `<GSI SecureMessage>`. The following subelements can be used to set various properties:

Element	Functionality
<integrity>	Sets protection level to signature
<privacy>	Sets protection level to encryption (signature is also done)
<peer-credential value=" <i>path to file with credentials to encrypt with</i> ">	Sets the path to the file containing the credential to use if privacy protection is chosen.

The following sample sets GSI Secure Message with integrity:

```

<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor
...
<GSI SecureMessage>
<integrity/>
</GSI SecureMessage>
...
</clientSecurityConfig>

```

6. GSI Secure Transport

The client can be configured to do GSI Secure Transport using the element <GSI SecureTransport>. The following subelements can be used to set various properties

Element	Functionality
<integrity>	Sets protection level to signature.
<privacy>	Sets protection level to encryption (signature is also done).
<anonymous>	Server is accessed as anonymous.

The following sample sets GSI Secure Transport with privacy and anonymous:

```

<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor
...
<GSI SecureTransport>
<privacy/>
<anonymous/>
</GSI SecureTransport>
...
</clientSecurityConfig>

```

7. Username/Password

Username/password can be used for authentication by the client. This is configured using <username> and <password-
Type> element. The username element allows for a string to be configured and the password configuration consists of a password and a type string.

Example configuration:

```
<clientSecurityConfig xmlns="http://www.globus.org/security/descriptor"
  <usernameType>
  <username value="tester1"/>
  <passwordType>
  <password value="TY^*(Hyu"/>
  <type value="someType"/>
  </passwordType>
  </usernameType>

</clientSecurityConfig>
```

8. Trusted credentials

Client side trusted credentials are configured similar to container security descriptor as described in [Section 11](#), “[Trusted Certificates](#)”. The outer element and schema for client security descriptor as described in [Section 1](#), “[Security Descriptor Schemas](#)” should be used.

If this configuration is not set, the underlying CoG JGlobus library is used to pick up trusted certificates. The library attempts to load the certificates as described in [Section 1](#), “[Trusted Certificates Location](#)” .

Chapter 7. Other configuration

1. Configuring Default GridMap File

The gridmap file is a common configuration in the toolkit and is typically configured within the GridmapPDP configuration. To specify a default value to be used across the toolkit, if not specified with in the GridmapPDP configuration, the `defaultAuthz` element in container security descriptor is used as described in [Section 5, “Default Authorization Chain”](#).

The gridmap authorization can be specified with any prefix, but the default configuration uses "gridmapAuthz" as shown in the example below.

Example:

```
<containerSecurityConfig xmlns="http://www.globus.org/security/descrip
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.globus.org/security/descriptor
name_value_type.xsd"
xmlns:param="http://www.globus.org/security/descriptor">

  <defaultAuthzParam>
    <interceptor name="gridmapAuthz:org.globus.wsrfl.impl.security.GridMapP
    <parameter>
      <param:nameValueParam>
        <param:parameter name="gridmap-file"
        value="/etc/grid-security/grid-mapfile"/>
      </param:nameValueParam>
    </parameter>
  </interceptor>
</defaultAuthzParam>
</containerSecurityConfig>
```

If the gridmap file is updated at runtime, it will be reloaded.

Glossary

G

grid map file

A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in `/etc/grid-security/grid-mapfile`. For more information see the Gridmap section [here](#).

P

proxy certificate

A short lived certificate issued using a EEC. A proxy certificate typically has the same effective subject as the EEC that issued it and can thus be used in its place. GSI uses proxy certificates for single sign on and delegation of rights to other entities.

For more information about types of proxy certificates and their compatibility in different versions of GT, see <http://dev.globus.org/wiki/Security/ProxyCertTypes>.

S

service credentials

The combination of a service certificate and its corresponding private key.